```
Out[1]:

Out[2]:
```

Click here to toggle on/off the Python code.

PHYS 10792: Introduction to Data Science

2019-2020 Academic Year

Course instructors: Rene Breton (http://www.renebreton.org) - Twitter @BretonRene (https://twitter.com/BretonRene)
Marco Gersabeck (http://www.hep.manchester.ac.uk/u/gersabec) - Twitter @MarcoGersabeck (https://twitter.com/MarcoGersabeck)

Chapter 12

Syllabus

1. Probabilities and interpretations
2. Probability distributions
3. Parameter estimation
4. Maximum likelihood + extended maximum likelihood
5. Least square, chi2, correlations
6. Monte Carlo basics
7. Probability
8. Hypothesis testing
9. Confidence level
10. Goodness of fit tests
11. Limit setting
12. **Introduction to multivariate analysis techniques**

Topics

12 Multi-variate analyses

12.1 Introduction

This part of the lecture follows chapter 5 of Behnke, written by Helge Voss. Some of the code examples are taken from www.scipy-lectures.org (http://www.scipy-lectures.org), which is a rich resource of examples with decent explanation in all aspects of scientific python from simple maths to machine learning.

The topic can easily fill a whole lecture course; hence, we have to be rather selective and to some extent superficial here.

There are two topics that are covered by multi-variate analyses: classification and regression. We will concentrate here on the former.

Classification covers the distinction of two or more classes of items in those of interest and those that are not.

The term multi-variate refers to the fact that we consider the information from several variables of interest, often called *features* in this context, rather than just analysing a single observable. The multi-variate analysis then transforms this set of variables into a single output variable. Let us call the input set $\mathbf{x} = \{x_1, \dots, x_D\}$, also referred to as the *feature vector*, and correspondingly the output $y(\mathbf{x})$.

You can think back to the Martian rock classification as an example. There, we discussed the classification based on a single oxide concentration. Within the context of this chapter, we can extend this to the joint analysis of all measured concentrations.

12.1.1 Linking classification to previous topics

Classification can be interpreted as a hypothesis test. In this case the null hypothesis, H_0, is that the item to classify is not of interest. Depending on the value of the test statistic $y(\mathbf{x})$ we then reject this hypothesis or not.

The performance of a multi-variate classifier is assessed by the rate of Type I and II errors, i.e. how often items are wrongly declared to be of interest (Type I error) or how often items are wrongly found to be not of interest (Type II error). As before the trade-off between these two rates depends on the application in question.

Let us call the items of interest signal (S) and those not of interest background (B). Their distinction is then linked to the test statistic exceeding a critical value in which case we accept the outcome as signal:

$$y(\mathbf{x}) > c.$$

We can consequently define a critical region C in parameter space where this equation is satisfied. With this, the rate of Type I errors or the significance is

$$\alpha = \int_C p(\mathbf{X}|H_0)d\mathbf{x} = \int_C p(\mathbf{X}|B)d\mathbf{x} = \int_{y(\mathbf{x})>c} p(\mathbf{X}|B)d\mathbf{x}.$$

This also defines the rate of Type II errors as

$$\beta = \int_{y(\mathbf{x})<c} p(\mathbf{X}|S)d\mathbf{x}.$$

In this context, we can also identify $1-\beta$, which is usually called power, as the signal efficiency, and $1-\alpha$ as the background rejection.

Rather than considering the complicated observable space in which we have a hyper-surface called decision boundary, defined by $y(\mathbf{x}) = c$, we can also consider the classifier output, which leads to

$$\int_c^\infty p(y|B)dy = \alpha,$$

and

$$\int_{-\infty}^c p(y|S)dy = \beta.$$

A graphical example is given below in 12.2.

Following Bayes, we can calculate the rate of signal decisions if we now the overall fraction of signal (f_S) and background (f_B) items as

$$P_S(y) \equiv P(S|y) = \frac{p(y|S)f_S}{p(y|S)f_S + p(y|B)(1-f_S)}.$$

12.1.2 Machine Learning

The term *machine learning* simply describes the automatic determination of the possible decision boundaries of a classifier, i.e. it determines the function $y(\mathbf{x})$ that transforms the feature vector into a single output quantity.

The term *supervised learning* refers to the usage of a training dataset for which the true classification is known. *Unsupervised learning* performs classification without being instructed which characteristics to pick out and is typically used in cluster finding or self-organising maps.

A classical example is the Iris, which a genus of 260-300 species of flowering plants. Some of the Iris varieties look very similar (see below) but have subtle differences when it come to the length and width of their petals for instance.

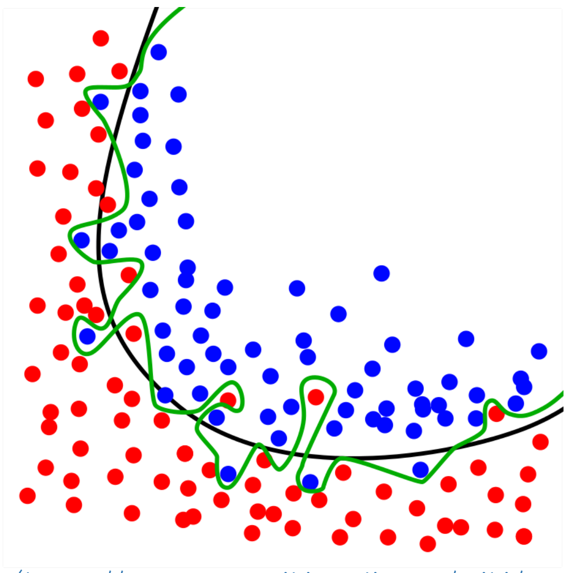
12.1.3 Bias-variance trade-off

When training a multi-variate classifier the configuration of the algorithm determines the number of degrees of freedom and through these the ability of the classifier to pick out small-scale features of the training dataset.

A classifier with few degrees of freedom will lead to very similar results when trained on statistically independent datasets that all are drawn from the same underlying distribution. This is called small *variance*. At the same time, the result will lead to a deviation from the individual dataset as it would not pick out the most fine-grained features; this is called the *bias* of the classifier.

Conversely, when the classifier is given many degrees of freedom, it will have a smaller bias at the expense of a larger variance. The balancing of these different criteria is called bias-variance trade-off.

At the point when it starts to pick out statistical fluctuations of the training set it is called *overtrained*. One way to avoid this is to exploit several datasets that all represent the same underlying distributions. They can for example all be randomly drawn subsets of one bigger sample. The first is called *training dataset* and is used for the initial round of machine learning. A second set, the *validation dataset* is used to evaluate the classifier. The training is performed with increasing complexity (i.e. number of degrees of freedom of the algorithm), which should lead to improving signal-background separation. So long as this separation is confirmed by the validation sample all is good. Once the validation sample no longer shows the same improvement of the training sample, the point of overtraining has been reached. Finally, the classifier is applied to a third sample, the test sample.



(Source: Chabacano (https://commons.wikimedia.org/wiki/User:Chabacano), Overfitting (https://commons.wikimedia.org/wiki/File:Overfitting.svg), CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/legalcode))

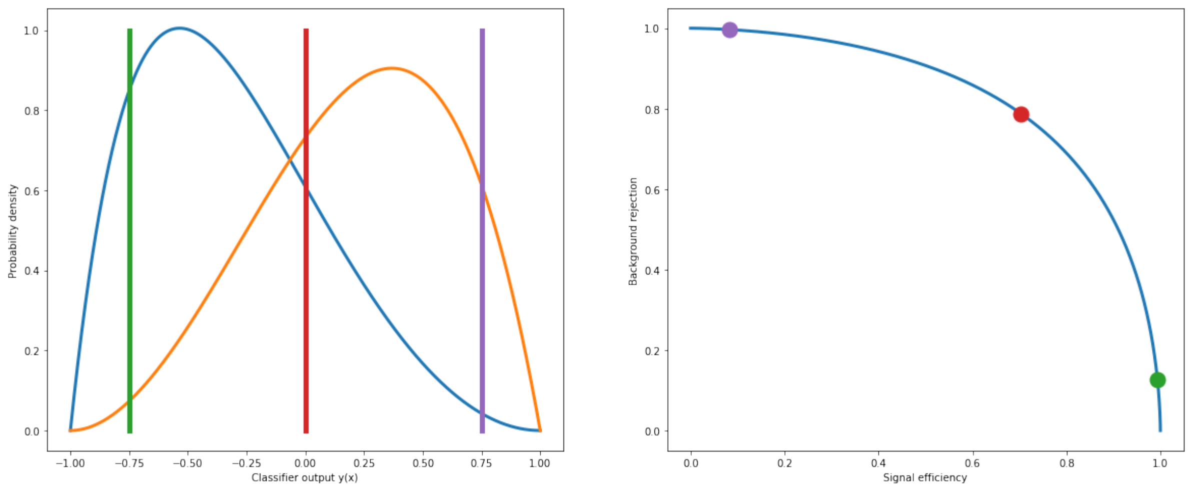
12.2 Receiver-Operator Characteristic curve

As discussed before, the optimal working point of Type I/II errors or signal efficiency and background rejection depends on the specific problem being analysed. It is instructive to illustrate the different options. The Receiver-Operating Characteristic (ROC) curve shows background rejection as a function of signal efficiency and is constructed by evaluating all possible critical values c.

The ROC curve must always start at $(0, 1)$, which corresponds to zero signal efficiency and 100% background rejection, i.e. everything is rejected. The opposite end of the curve is at $(1, 0)$, where everything is accepted, i.e. 100% signal efficiency but zero background rejection.

The example below uses two hypothetical signal and background classifier output distributions and builds a ROC curve for these, including highlighting some example points.

The accepted region lies to the right of the threshold value. The orange curve represents the signal distribution and the blue curve shows the background distribution. The right-hand graph shows background rejection against signal efficiency, i.e. the ROC curve.

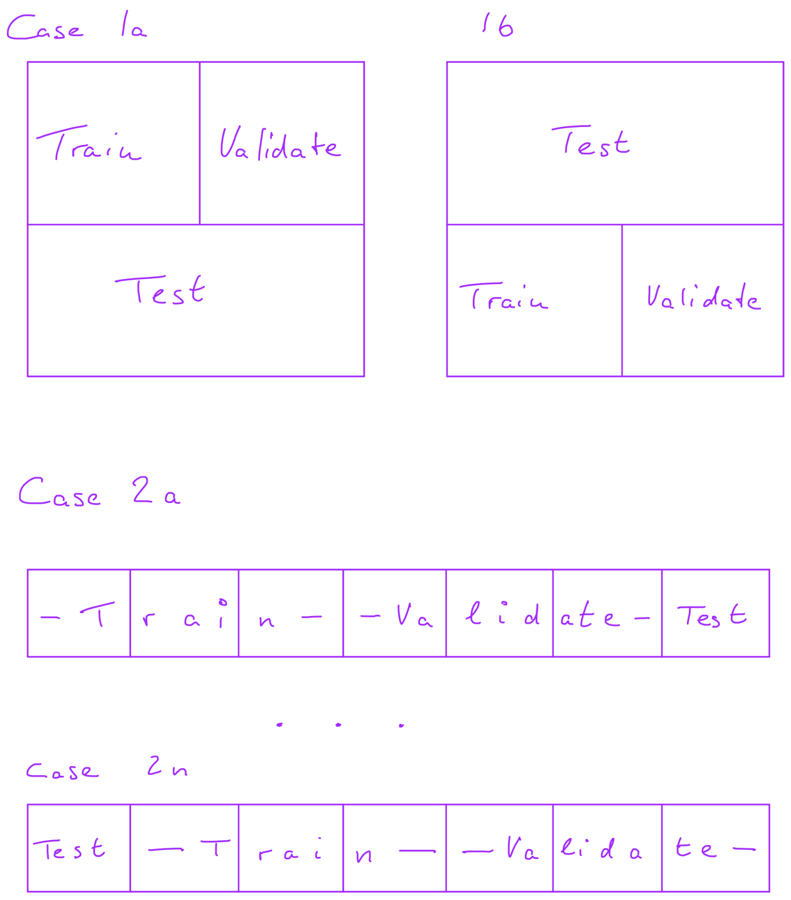


12.3 Cross-validation

Cross-validation describes a useful approach when the sample size is limited. It refers to a given dataset being split in e.g. ten subsets. If training and validation are carried out on four of the ten subsets each, they can exploit nearly all of the statistical power of the data. The classifier is then applied to the remaining two subsets.

The assignment of the ten subsets to the three samples, training, validation and test, are cycled through, such that (in this example) five classifier trainings have to be carried out. While this comes at some computational expense is has the advantage of making optimal use of the available data without having to sacrifice a subset for training and validation.

The plot below illustrates two scenarios of splitting a dataset into training, validation, and test datasets.



Bootstrapping

Before diving into different concrete techniques, it is a good point to introduce a concept that is very powerful and widely used: bootstrapping.

One is often faced with datasets of limited size but the task of creating an ensemble of datasets to be used in the context of hypothesis testing (which includes classification).

Such ensembles can be created via bootstrapping in two different ways depending on the exact context:

- One can randomly draw a subset from the full dataset (i.e. by drawing with replacement) and repeat this multiple times. This generates different subsets; however, they are not completely statistically independent.
- In the context of classification one can randomly assign signal and background labels (or whatever categories apply) to a dataset and repeat this to generate multiple datasets with randomised labels.

Both variants can be used for example to create an ensemble of datasets to establish the distribution of a test statistic for a null hypothesis from.

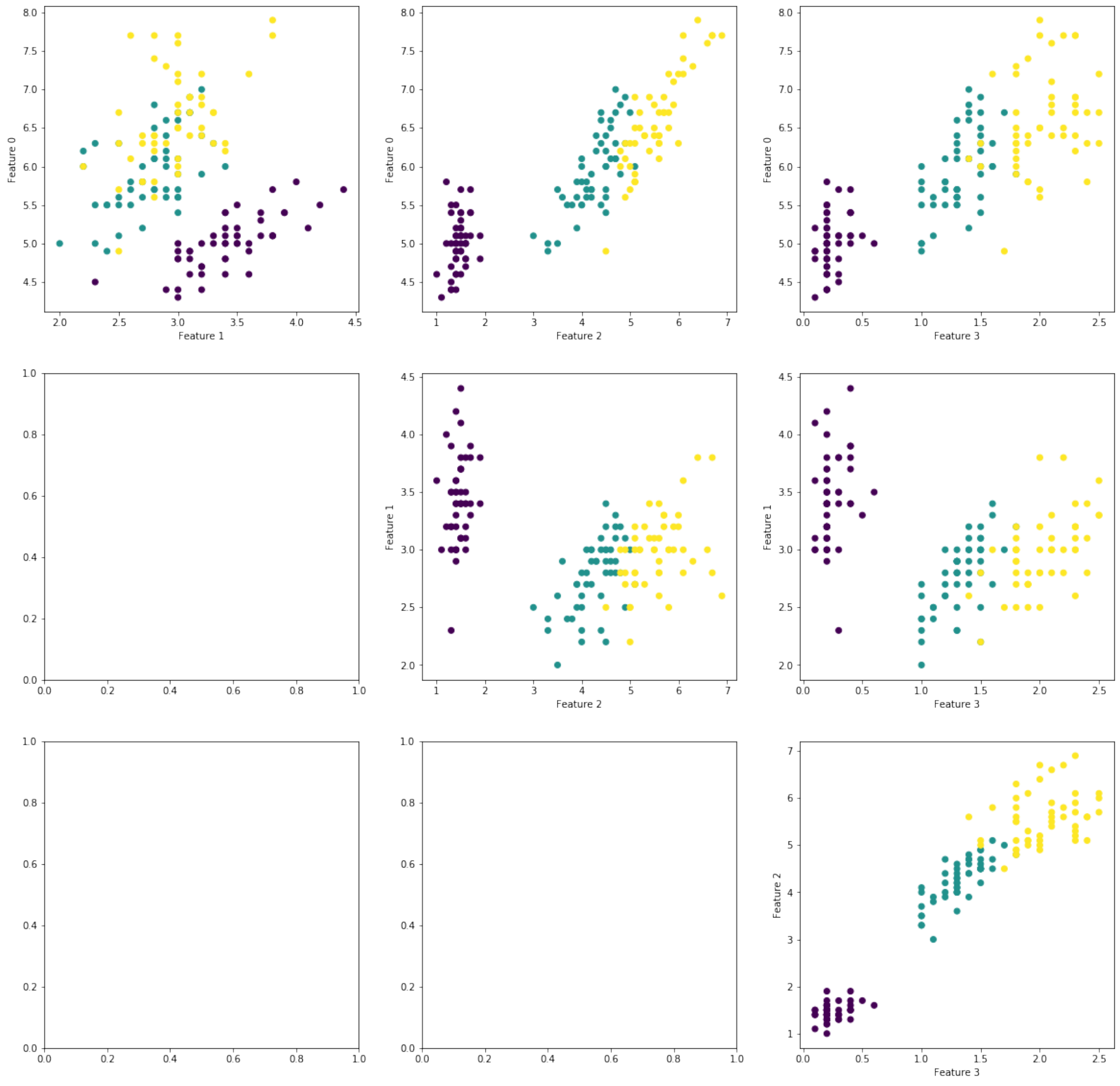
Multi-variate classification techniques

In the following a few classification techniques are introduced. In the context of this lecture, this can only happen at a rather superficial level. Anyone interested in this is referred to the vast amount of literature available on this topic, which of course includes the two resources used for this lecture.

- Multi-variate classification techniques
  - Likelihood
  - k-Nearest Neighbour
  - Artificial Neural Networks
  - Boosted Decision Trees

Some input data

```
(150, 4)
[5.1 3.5 1.4 0.2]
(150,)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['setosa' 'versicolor' 'virginica']
```



k-Nearest Neighbour Classifier

The power of multi-variate classifiers is that they can pick out features that are not obvious by looking at individual observables. This is where the "curse of dimensionality" enters as a small fraction of input phase space represents a large fraction of each variable. As an example, considering a ten-dimensional hypercube of size 1^{10}, a sub-region that covers as much as half the length in each dimension covers about 0.1% of the total volume.

The k-Nearest Neighbour (kNN) approach addresses this by counting the number of items of the same class (signal or background) in a hyper-sphere around the event in question. By counting up to a total of k nearest neighbours, the method automatically scales the size of the volume that is investigated with the density of entries.

The local densities can then be approximated by the number of signal and background items found among the k nearest neighbours as

$$\frac{p(\mathbf{x}|S)}{p(\mathbf{x}|B)} \propto \frac{P(S|\mathbf{x})}{P(B|\mathbf{x})} \approx \frac{k_s(\mathbf{x})}{k_b(\mathbf{x})}.$$

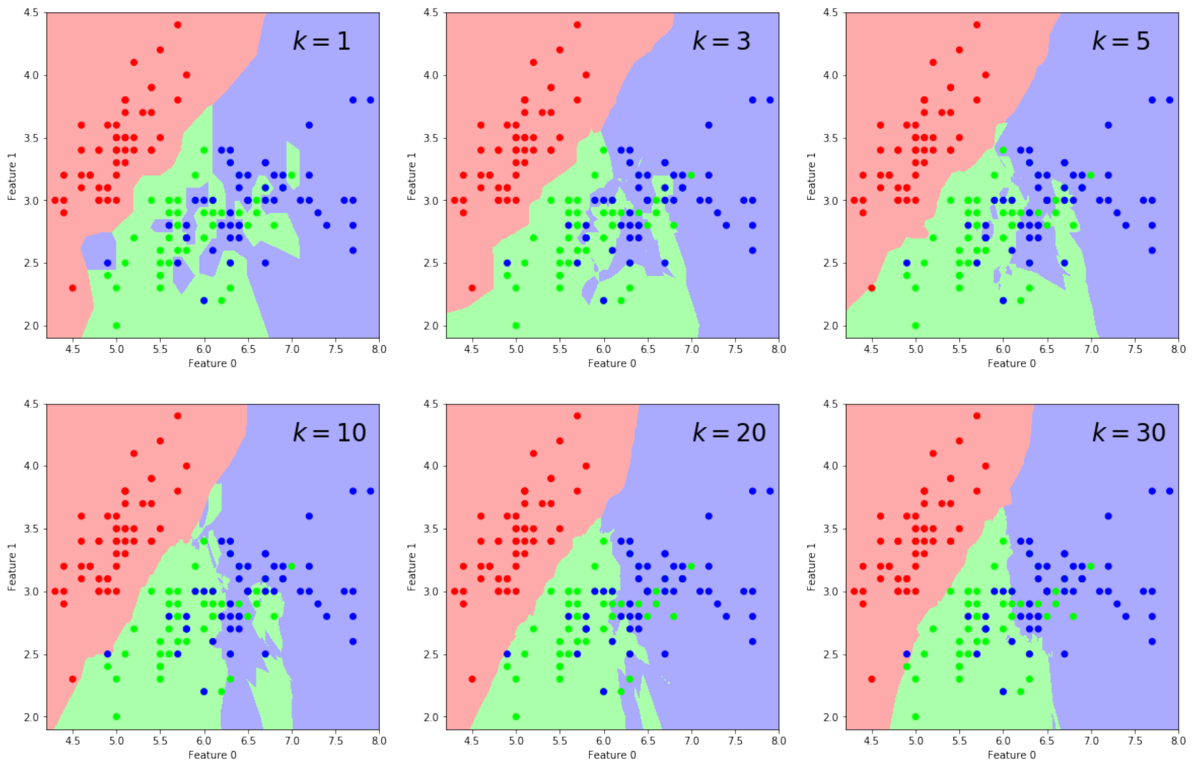
Taking the numbers of signal and background events in the training sample, one can define the probability of a test event at the phase-space point $\mathbf{x}$ being of signal type:

$$P_s(\mathbf{x}) = \frac{k_s(\mathbf{x})}{k_s(\mathbf{x}) + k_b(\mathbf{x})} = \frac{k_s(\mathbf{x})}{k(\mathbf{x})}.$$

The example below shows four-dimensional data analysed by a kNN classifier. It shows all possible 2D projections and the classifier output for a range of values of k.

```
['virginica']
```

```
Out[6]: Text(7,4.2,'$k=30$')
```



Comments

We can see that taking into account more and more neighbours leads to smoother contours of the classification boundaries. The $k = 1$ case almost certainly represents a severe case of overtraining as it picks out almost all of the features of the training sample.

For a full interpretation of the quality of the training, we would require a validation sample. Note that the plots above represent a 2D view of a 4D parameter space.

Artificial Neural Networks

Artificial neural networks use the combination of an arbitrary number of functions to pick out features in the dataset. They are based on a linear mapping

$$y(\mathbf{x}) = w_0 + \sum_{m=1}^{M} w_m \cdot h_m(\mathbf{x}),$$

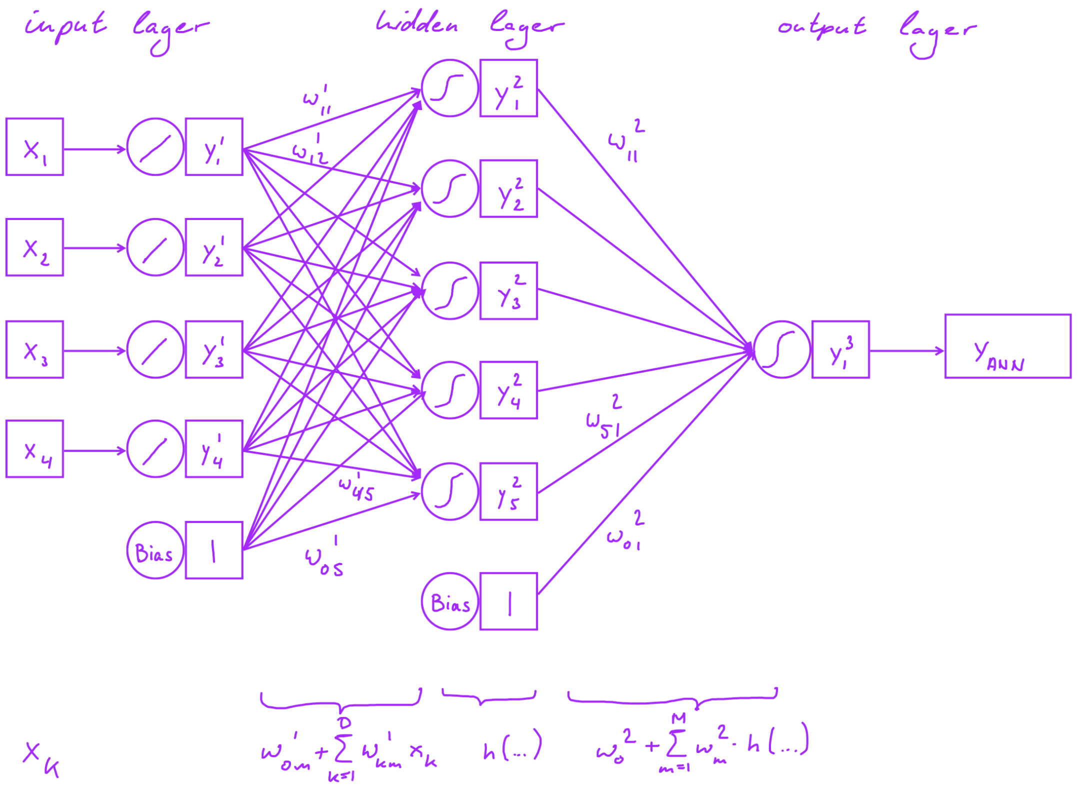
with weights w_i and basis functions $h_m(\mathbf{x})$.

It is more efficient to use a single function $h(t)$ for which functions like sigmoid, $h(t) = 1/(1 + e^{-t})$, turn out to be most useful as they are approximately linear for some variable range and constant otherwise. This means they select or "activate" a part of the variable range and are therefore called activation function.

In order to retain flexibility when using a single function, the variables undergo a linear transformation resulting in

$$y(\mathbf{x}) = w_0^2 + \sum_{m=1}^{M} \left[w_m^2 \cdot h\left(w_{0m}^1 + \sum_{k=1}^{D} w_{km}^1 x_k \right)\right].$$

The following plot shows the schematic layout of such a network with $D = 4$ input variables and $M = 5$ activation functions.



Several different structures of neural networks exist with increasingly complex structures, e.g. with more than one hidden layer.

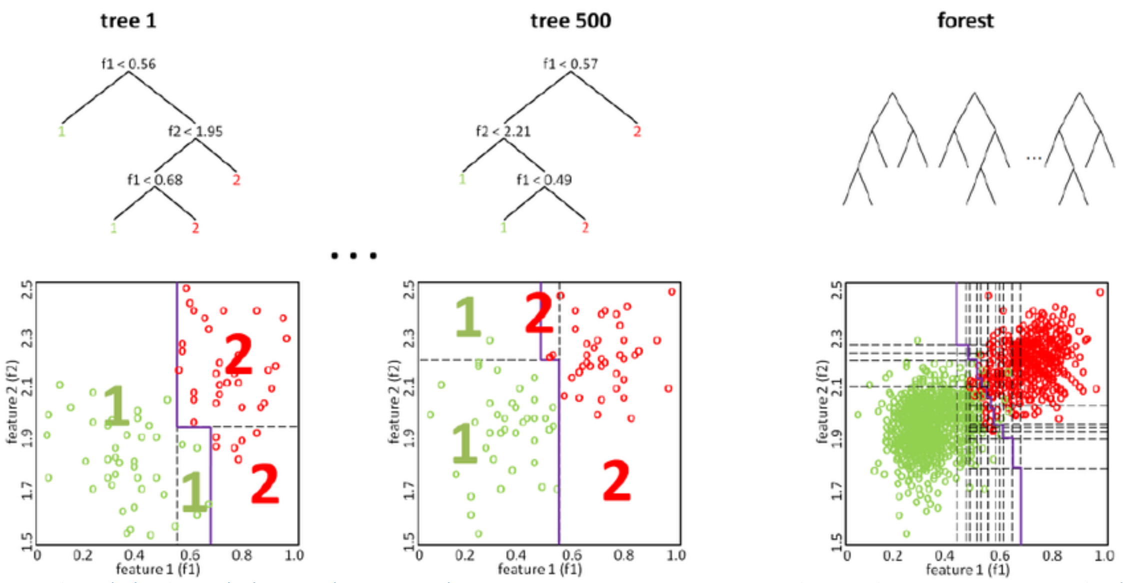
(Boosted) Decision Trees

Decision trees are based on a tree-like structure in which a decision is taken based on one variable and each of the outcomes then leads on to a decision being taken for the next variable, where the second level depends on the outcome of the first.

As a single decision tree offers limited flexibility, there are a number of concepts that increase complexity. One very widely used technique is boosting. This uses the output of several decision trees and computes a linear combination thereof to achieve a classifier output

$$y(\mathbf{x}; \alpha_0, \ldots, \alpha_M, \mathbf{a_0}, \ldots, \mathbf{a_M}) = \sum_{m=0}^{M} \alpha_m b(\mathbf{x}; \mathbf{a_m}),$$

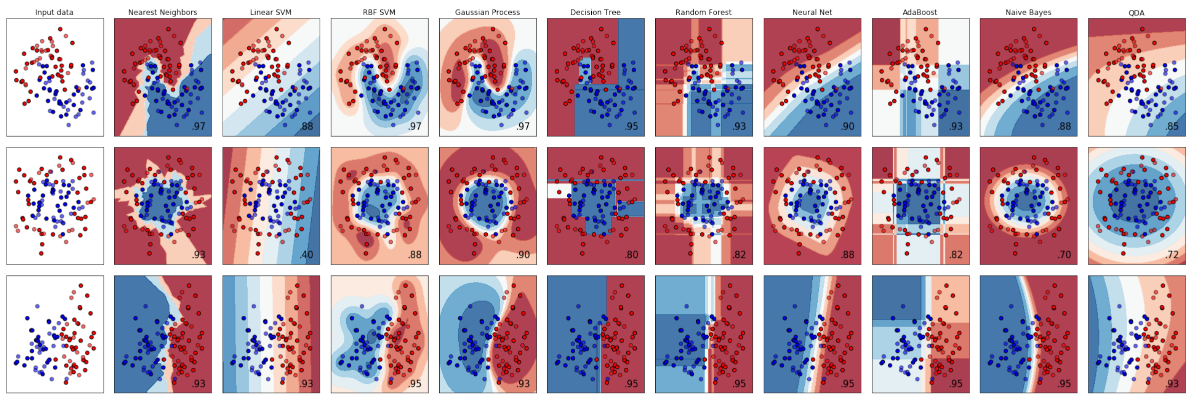
where $\mathbf{a_m}$ are the parameters of the individual trees and α_m are the boosting weights. There are different algorithms to optimise the boosting and train these classifiers.



(Source: [Towards Digital Staining using Imaging Mass Spectrometry and Random Forests-Technical Report (https://www.researchgate.net/publication/228540194_Towards_Digital_Staining_using_Imaging_Mass_Spec Technical_Report)](https://www.researchgate.net/publication/228540194_Towards_Digital_Staining_using_Imaging_Mass_Spec), Michael Hanselmann et al., 2009)

```
Automatically created module for IPython interactive environment

/usr/local/lib/python3.7/site-packages/sklearn/neural_network/multilayer_percep
tron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200)
reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/site-packages/sklearn/neural_network/multilayer_percep
tron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200)
reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/site-packages/sklearn/neural_network/multilayer_percep
tron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200)
reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```


Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
Random Forest
Neural Net
AdaBoost
Naive Bayes
QDA